E-24-656

APPENDIX VII

NATIONAL SCIENCE FOUNDATION
Washington, D.C. 20550

FINAL PROJECT REPORT
NSF FORM 98A

PLEASE READ INSTRUCTIONS ON REVERSE BEFORE COMPLETING

PART I–PROJECT IDENTIFICATION INFORMATION

1. Institution and Address	2. NSF Program	3. NSF Award Number
School of Industrial and Systems Eng. Georgia Institute of Technology Atlanta, GA 30332	System theory and Operations Research	ECS-8307230
	4. Award Period From 6/15/83 To 11/30/85	5. Cumulative Award Amount 47,993

6. Project Title

Research Initiation: Sensitivity Analysis and Rescheduling Algorithms for One-Stage Scheduling Problems

PART II–SUMMARY OF COMPLETED PROJECT *(FOR PUBLIC USE)*

Real scheduling problems are always changing because of the dynamic nature of production environments. The goal of the project is to study the sensitivity of scheduling problems to changes, and to develop rescheduling algorithms which solve a given problem given the solution to a slightly altered problem. Attention is restricted to single stage problems (i.e. no flowshops or jobshops) with single or parallel processors.

For hard problems such as the travelling salesman problem or makespan minimization on parallel machines, it was found that the sensitivity to small changes in data was severe. For example, the regions of optimality for the planar travelling salesman problem do not have nice properties such as convexity or connectedness. It was shown that one cannot find the new optimal solution to a slightly altered problem by iteratively making slight improving changes to the old solution. The best approach seems to be to develop heuristics which are so fast they can be run in real time. This was done for a minimum waste sequencing problem arising in satellite data analysis.

For problems solvable in polynomial time, rescheduling algorithms offer some benefits. In general, it appears that with the exception of sorting-based algorithms, the worst-case performance of the rescheduling algorithm will be the same as for the scheduling algorithm, but the average performance will be better. This is true of problems solved by linear programming or network methods, and also dynamic programming-based methods. The rescheduling research also led to a general method to improve augmenting path algorithms by layering.

PART III–TECHNICAL INFORMATION *(FOR PROGRAM MANAGEMENT USES)*

1. ITEM *(Check appropriate blocks)*	NONE	ATTACHED	PREVIOUSLY FURNISHED	TO BE FURNISHED SEPARATELY TO PROGRAM	
				Check (√)	Approx. Date
a. Abstracts of Theses	✓				
b. Publication Citations		✓			
c. Data on Scientific Collaborators		✓			
d. Information on Inventions	✓				
e. Technical Description of Project and Results		✓			
f. Other *(specify)*					

2. Principal Investigator/Project Director Name *(Typed)*	3. Principal Investigator/Project Director Signature	4. Date
Craig A. Tovey		6/9/86

NSF Form 98A (3-83) Supersedes All Previous Editions

Form Approved OMB No. 3145-0058

Publication Citations

"Rescheduling to Minimize Makespan on a Changing Number of Identical Processors," Naval Research Logistics Quarterly 33, 1986.

"Multiple Optima in Local Search," to appear in Journal of Algorithms (with V. Rodl).

"Layered Augmenting Path Algorithms," Mathematics of Operations Research 11, No. 2, pp. 362-370, 1986 (with E. Tardos and M. Trick).

"Performance Bounds on Machine Configuration in Manufacturing Systems," Technical Report MHRC-OP-85-01, Georgia Institute of Technology, 1985 (with C. Lofgren).

"Minimum Spillage Sequencing," submitted to Management Science (with G. Weiss and J. Wilson).

Data on Scientific Collaborators:

Christopher Coleman : graduate student

Technical Summary

Let T be a permutation of the integers 1,...,n. Treating T as a tour on n cities for a planar travelling salesman problem, we define the region of optimality F(T) to be the region in the plane where city n can be placed, leaving the other cities fixed, such that T is optimal. Of course, T can be the empty set. The question addressed is, what do the regions F(T) look like?

The answer is that F(T) is in general neither convex nor connected. (See diagrams in attachment 3). This is true even for the small case n = 4. We conjecture that F(T) is simple (i.e. for any closed curve in F(T), the interior of that curve is also in F(T)). We proved the conjecture for n=4 and proved that the regions of local optimality, F'(T), are simple. The region of local optimality is defined as is F(T) except that T need only be locally optimal with respect to 2-opting in the region F'(T).

We found that the regions of optimality for the related but easier problem of finding a minimum spanning tree do have some nice properties, but these results duplicated results of Papadimitriou (1985).

For the problem of minimizing makespan on m parallel identical processors when m changes value, the optimal solution is less robust than the LPT heuristic solution. The rescheduling problem in this case is NP-hard. One implication of this result is that, unless NP = Co(NP), it is not possible to solve the "new" problem by applying a local improvement procedure to the "old" solution (see attachment 1). In fact, there exist instances of the problem so that any such procedure, no matter what the definition of the neighborhood, can get stuck in exponentially many places (see attachment 2).

The large amount of senstivity displayed by these and other problems indicates that optimizing rescheduling algorithms are not an appropriate goal for these problems. The best approach seems to be to develop heuristics which run in real time. If the problem changes, the scheduling heuristic can simply be run again at negligible time cost to determine the new schedule. This approach was taken for a minimum waste sequencing problem arising from satellite data processing (see attachment 4). It was interesting that consideration of the dynamic case, where jobs arrive during the processing time, led to the best algorithm for the static case.

A cardinality constrained version of the makespan minimization problem was also considered. The sensitivity is at least as bad, since the unconstrained problem is a special case. The problem arises in flexible manufacturing systems, an area notorious for dynamic environmental conditions such as machine breakdown, rush orders, etc.

The LPT-based heuristic developed and analyzed is very fast, asymptotically optimal, and obviously allows rescheduling in linear time (see attachment 5).

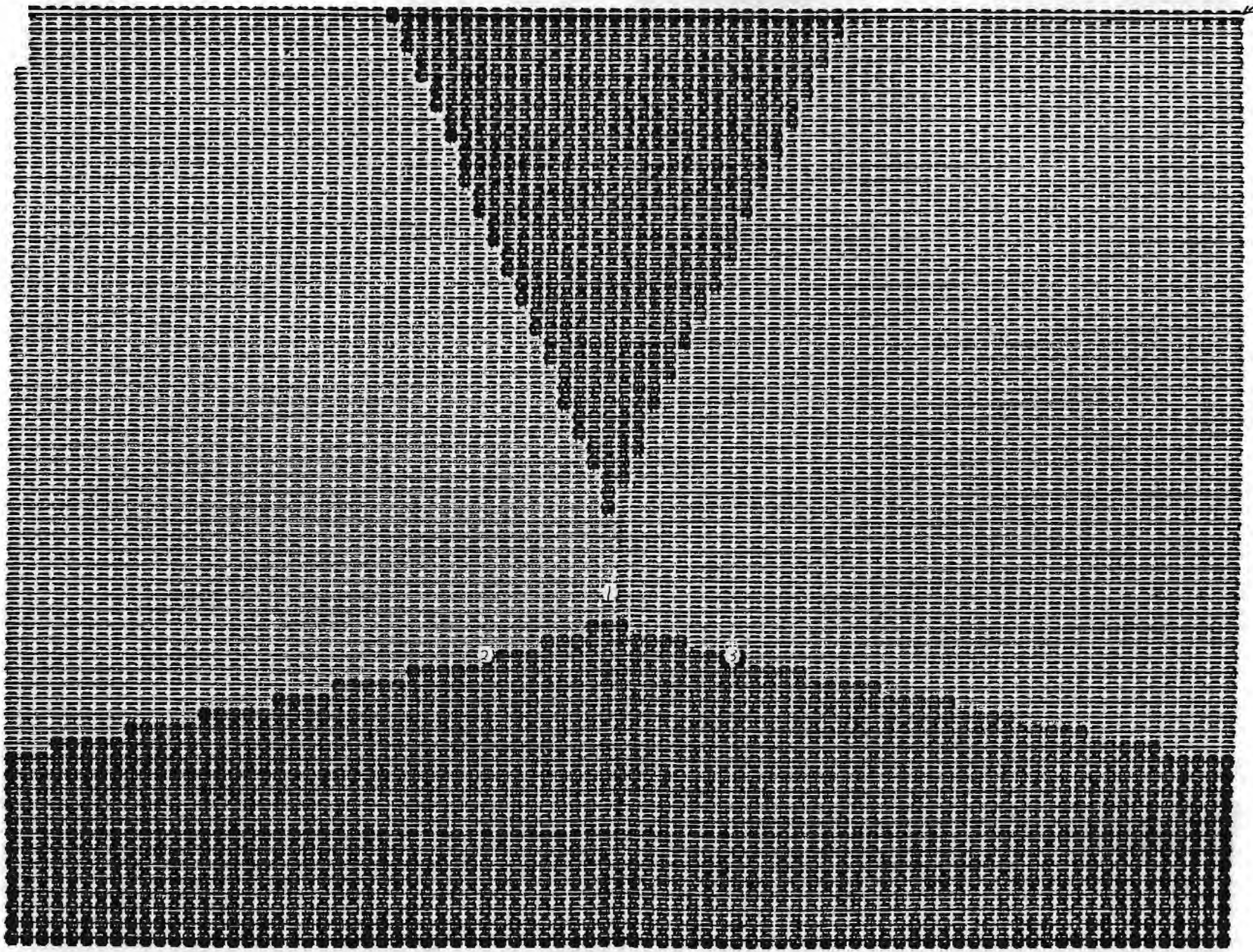
Investigation of the possibility of efficiently rescheduling problems with due dates and release times led to the question of the existence of a layered version of the Lawler-Martel polymatroid flow algorithm. This led to a general method for improving the efficiency of augmenting path algorithms (see attachment 6).

ATTACHMENT 1

ATTACHMENT 2

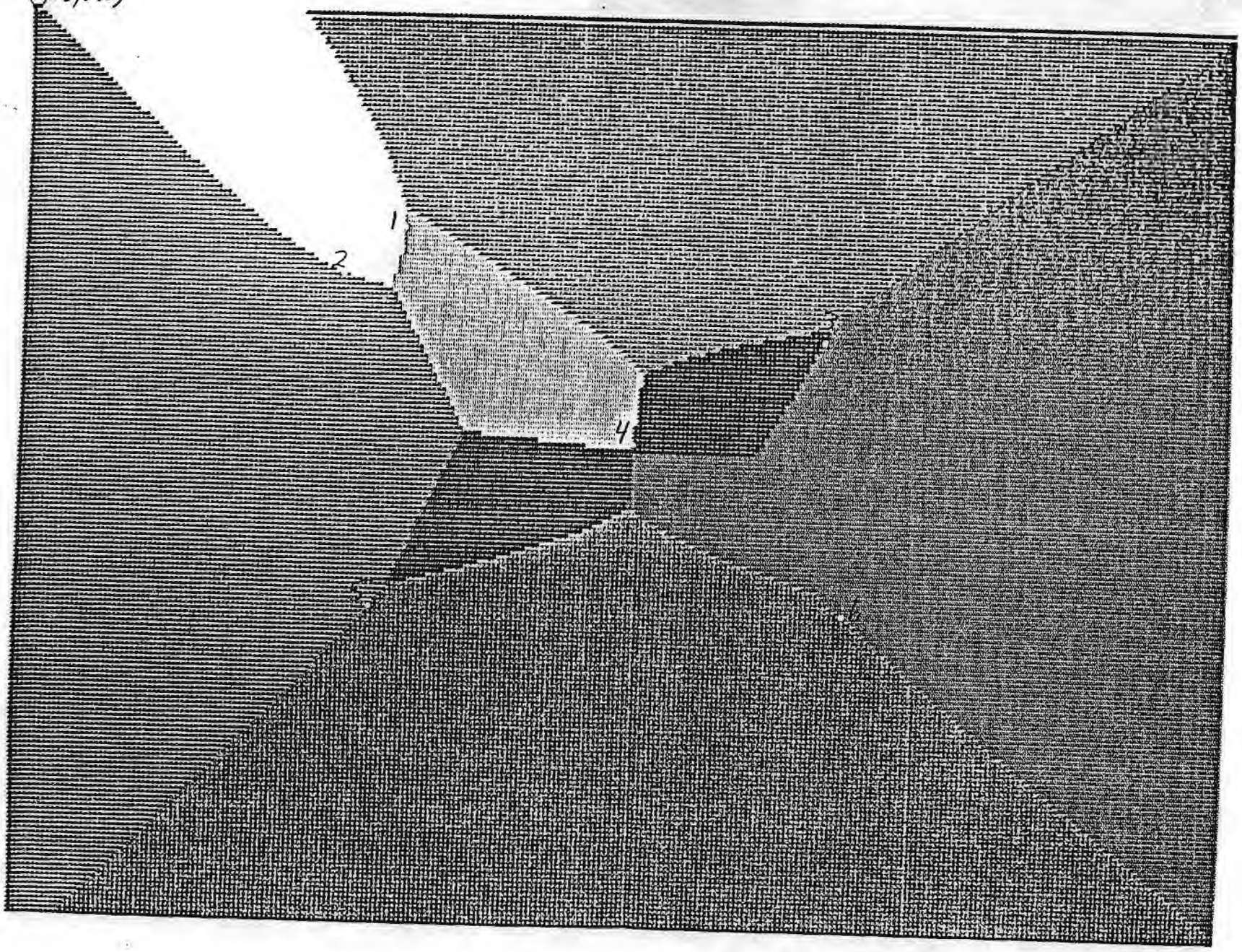
ATTACHMENT 3

flat triangle
(500, 470)
(400, 520)
(600, 520)
resolution = 6



(320,100)
(270,220)
(680,270)
(520,360)
(300,500)
(700,500)

res = 2



ATTACHMENT 4

ATTACHMENT 5

ATTACHMENT 6

Layered Augmenting Path Algorithms

Éva Tardos[†]

Craig A. Tovey[††]

Michael A. Trick[††]

[†]Institut fur Operations Research, Bonn. W. Germany

[††]School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332

Research supported by NSF Grant Number ECS-8307230.

To appear in Math OR

Abstract

Augmenting path algorithms, first introduced by Ford and Fulkerson [7], are widely used in optimization. Examples include Schönsleben's polymatroid intersection algorithm [15], the maximum polymatroidal network flow algorithm of Lawler and Martel [11], Frank's algorithm for the Edmonds-Giles polyhedron [8] and Cunningham's algorithm for testing membership in matroid polyhedra [2].

Here we give an order of magnitude improvement for the above algorithms by using an approach analogous to that of Dinits' maximum flow algorithm [4].

1.0 Introduction

Augmenting path algorithms, first introduced by Ford and Fulkerson [7], are widely used in optimization. Examples include the maximum polymatroidal flow algorithm of Lawler and Martel [11], Schönsleben's polymatroid intersection algorithm [15], Frank's algorithm for the Edmonds-Giles polyhedron [8] and Cunningham's algorithm for membership in matroid polyhedra [2].

Dinits [4] and Edmonds and Karp [6] modified the Ford-Fulkerson algorithm to improve its efficiency. In this paper we show how Dinits' idea can be used to improve the efficiency of the above algorithms. We give a formal definition of an augmenting path algorithm and state sufficient conditions to guarantee the existence of a more efficient layered (Dinits-like) version. As corollaries we deduce an $O(m^4d)$ algorithm for polymatroid network flows (an improvement over $O(m^5d)$ in [11]) and an $O(n^4h)$ algorithm for finding feasible vectors in the Edmonds-Giles polyhedron (versus $O(n^5h)$ in [8]).

Further improvements on the flow algorithm [1,10,13] employ the same layered graph. However, the analogy of the flow algorithm to these augmenting paths algorithms is imperfect, and improvements for these algorithms do not follow directly.

2.0 Definition of Augmenting Path Algorithms

Augmenting path algorithms can be used for many types of problems, not just "max-flow" algorithms. A problem consists of a set of instances where the objective in each instance is to find a vector x from a set of vectors S feasible for the instance. The vector to be found might optimize some function f(x), it might satisfy some feasibility conditions, or it might satisfy other conditions.

An augmenting path algorithm for a problem associates with each instance of the problem a directed graph $H = (N,A)$ where N is the node set and A is the arc set (multiple arcs are allowed). It has two distinguished nodes s and t. Associated with each $x \in S$ is a nonnegative capacity function $\delta_x(i,j)$ for each arc $(i,j) \in A$. Let $H_x = (N,A_x)$ be a capacitated directed graph with capacities δ_x and $A_x = \{(i,j) \in A: \delta_x(i,j) > 0\}$. A directed path from s to t in H_x is called an s-t augmenting path.

The form of an augmenting path algorithm is

Algorithm 2.1 Augmenting Path Algorithm

Step 0 (initialization) Set $x \leftarrow x_0$, an initial feasible solution.

Step 1 (calculation H_x) Find the capacitated directed graph H_x using δ_x.

Step 2 (find augmenting path) Find an s-t augmenting path P in H_x.
If no such P exists, go to Step 4.
Otherwise proceed to Step 3.

Step 3 (augment solution) Replace $x \leftarrow x'$ where x' is a new solution defined by x and P.
Go to Step 1.

Step 4 (terminate) Stop, the current x is optimal.

For example, the Ford-Fulkerson algorithm to solve the maximum flow problem on a network with grah $G = (V,E)$ is an augmenting path algorithm with $N = V$ and $A = E \quad E^{-1}$ (where E^{-1} is the set created by reversing the arcs of E). The function $\delta_x(i,j)$ is the excess capacity on arc (i,j) for $(i,j) \in E$ and $\delta_x(i,j)$ is the flow on (i,j) for $(i,j) \in E^{-1}$. The augmented

flow x' is found by finding the minimum $\delta_x(i,j)$ for (i,j) on P and adding this value to the current flow (if $(i,j)\varepsilon E$) or subtracting it (if $(i,j)\varepsilon E^{-1}$) for all arcs on P. The validity of Step 4 follows from the minimum cut which can be extracted at Step 2.

The fundamental property all augmenting path algorithms must possess is the validity of the termination step. This property can be written

Property 0: An $x\varepsilon S$ is optimal if and only if there does not exist an s-t augmenting path in H_x.

An algorithm that satisfies only property 0 may be a very poor algorithm; it may not terminate in finite time or may not even converge to an optimal solution. We can write the time complexity of an augmenting path algorithm as follows:

define

n: $|N|$

a: $|A|$

d: amount of time to calculate $\delta_x(i,j)$ for $(i,j)\varepsilon A$

u: amount of time to update x to x'

t: number of solutions examined before optimality.

Then an augmenting path algorithm has time complexity $O(t(ad+u))$.

The following sections give a method of decreasing this value, first bounding t by choosing specific s-t augmenting paths and then using a "layered network" to decrease the amount of work.

3.0 Sufficient Conditions for Layering

To motivate our conditions sufficient to guarantee the existence of a layered algorithm, we briefly discuss the complexity of the Dinits and Edmonds-Karp modifications of Ford-Fulkerson's maximum flow algorithm.

For the current solution x in an augmenting path algorithm, let

$\sigma_x(i) \equiv$ the length of a shortest s-i augmenting path in H_x for $i \in N$.

If no such path exists, let $\sigma_x(i) = \infty$.

and $\tau_x(i) \equiv$ the length of a shortest i-t augmenting path in H_x for $i \in N$.

If no such path exists, let $\tau_x(i) = \infty$.

Define the following layered graph D_x.

$$D_x = (N, F_x) \text{ where } F_x = \{(i,j);\ \delta_x(i,j) > 0,\ \sigma_x(i) = \sigma_x(j) - 1\}$$

The k^{th} layer in D_x is the set of nodes i such that $\sigma(i) = k$. By definition, arcs in F_x link vertices in adjacent layers.

Edmonds and Karp [6] observed that if, in Step 2, P is chosen to be as short as possible, then all edges (i,j) in $A_{x'}$ but not in A_x (i.e., $\delta_x(i,j) = 0$ and $\delta_{x'}(i,j) > 0$) have $\sigma_x(j) < \sigma_x(i)$. Thus $\sigma_x(i)$ and $\tau_x(i)$ are non-decreasing for all i in N.

The algorithm then naturally divides into phases. During phase p, $\sigma_x(t) = p$; thereafter $\sigma_x(t) > p$. Since a path cannot have length more than $|N|$, the complexity of the algorithm is bounded by $r \cdot n$, where r is the work required by one phase.

Moreover, by the same observation, the edge set of the layered graph D_x is non-increasing during a phase. It is strictly decreasing when an augmenting path is found (the edges with minimum δ_x along the path are deleted). So no more than a augmenting paths can be found during a phase, and thus $r = a^2$.

Now Dinits' improvement is the following: Construct the layered graph D_x at the beginning of each phase. By the above observation of Edmonds and Karp updating D_x in a phase takes only O(n) steps. Moreover,

as D_x is nonincreasing in the phase, all edges of D_x from which node t is not reachable will not be used any more in this phase, so they can be deleted.

Now use depth first search on D_x. After at most O(n) steps an edge will be deleted, because either the node t is not reachable from that edge (this happens if the depth first search steps back from that edge) or an augmenting path is found and the edges with σ_x minimal along the path are deleted. Thus a phase of Dinits' algorithm requires only O(na) work.

Next we show how this generalizes to other augmenting path algorithms. The problem is that Edmonds-Karp's observation is not valid for the examples mentioned in the introduction. Even if the augmenting path is chosen to be minimal, we can only show that $\sigma_x(j) \leq \sigma_x(i)+1$ for an edge (i,j) in $A_{x'}$ and not in A_x. This also implies that the algorithms have

Property 1: If the augmenting paths in Step 2 are chosen to be the shortest s-t augmenting paths then $\sigma_x(i)$ and $\tau_x(i)$ are non-decreasing for all i.

The other important property of the Edmonds-Karp algorithm was

Property 2: If all augmentations use minimum length s-t augmenting paths, then after an augmentation changing x to x', the first arc on the path with $\delta_x(i,j)$ minimal will have $\delta_{x'}(i,j) = 0$.

Property 1 enables us to divide the algorithm into phases according to the value $\sigma_x(t)$. But, as there are new edges appearing in the layered graph, these properties do not seem to be sufficient to bound the time required by a phase. In the augmenting path algorithms mentioned in the introduction it is possible to place a further restriction on how an arc

can change from zero to positive capacity. Then with a special choice of the augmenting paths, the time required by a phase can be bounded. To do this it is shown that the algorithms have the following Property 3.

First we need to define a lexicographic ordering on the paths.

Definition. Index the nodes of H arbitrarily. Given two paths P and P' with the same number of arcs, we say that P is lexicographically smaller than P', written $P \lessdot P'$, if the vector of the nodes in P, placed in reverse order,is lexicographically smaller than the vector of nodes in P' in reverse order.

Let y be the feasible solution examined when a new phase starts.

Property 3: If augmentations are made along the lexicographically minimal (in short: lex-min) shortest paths then the paths that realize $\sigma_y(i)$ are lexicographically non-decreasing during a phase for each node i.

Lemma 3.1 For an augmenting path algorithm that augments along the lex-min shortest augmenting paths and satisfies properties 0-3 the number of solutions examined in a phase is at most a.

Proof: Let x be a solution examined and P be the augmenting path used. By the choice of P, all its s-i subpaths are lexicographically minimal in H_x. Let (i,j) be the first edge of P with $\delta_x(i,j)$ minimal. By Properties 2 and 3 the lex-min shortest path from s to j after the augmentation is lexicographically greater than the s-j subpath of P. Thus, by Property 3 again, the edge (i,j) will be ineligible for the rest of the phase.

After at most a augmentations all edges will be ineligible so there can be no more than a augmentations per phase. ∎

Another property is closely related to Property 3. Let $\pi_x(i)$ denote the node j such that (j,i) is the lexicographically minimal arc with $\delta_x(j,i) > 0$, if one exists; let $\pi_x(i) = +\infty$ otherwise.

Lemma 3.2 If augmentations are done on lexicographically minimal shortest augmenting paths and if, for every augmentation x to x' within a phase, $\pi_x(i) \geq \pi_{x'}(i)$ for all nodes i, then property 3 is satisfied.

Proof: Apply the assumption of the lemma inductively.

Now the total complexity of an augmenting path algorithm which satisfies properties 0-3 is $O(na^2d+nau)$, since there are at most n phases and so $t \leq na$.

In the next section, we improve the running time of these augmenting path algorithms by giving a version which uses layered graphs. The section also includes a proof of its correctness.

4.0 The Layered Network Algorithm

Properties 0 through 3 allow us to decrease the effort required in each phase by use of a layered graph L_p. The main algorithm is Algorithm 4.1.

Algorithm 4.1 Main Algorithm

```
Begin
    For p = 1 to n do
        begin
            Create L_p using Algorithm 4.2
            Update x using Algorithm 4.3 with L_p
```

end;

end.

Algorithm 4.2 Create L_p

Input: the solution on entering phase p: x.

Output: L_p.

Step 1 (calculate σ and τ) Calculate $\sigma_x(i)$ and $\tau_x(i)$ for all i using $\delta_x(i,j)$.

Step 2 (place nodes) Place a node labelled i in layer ℓ of L_p if and only if $\sigma(i) = \ell$ and $\sigma(i) + \tau(i) = p$.

Step 3 (place arcs) Add an arc between i and j in L_p if and only if $(i,j) \in A$ and $\sigma(i) = \sigma(j)-1$.

Step 4 (terminate) Stop.

Note that the layered network created is different from Dinits'. Dinits places an arc (i,j) in L_p only if $\delta_x(i,j)>0$. Though suitable for pure network flows, this requirement is too restrictive in general. The reason is that an arc's capacity may increase from zero during a phase and the arc may then appear in a shortest augmenting path during the same phase. As we verify in Theorem 4.1, any such arc will have to satisfy the requirements of Step 3 at the onset of the phase and so will be contained in L_p.

Algorithm 4.3 Find Solutions in L_p

Input: L_p; the current solution x.

Output: a solution x such that $\sigma_x(t) > p$.

Step 0 (initialize) Mark all edges "unblocked".

Step 1 (find the augmenting path) Use depth first search starting at node t to find the lex-min shortest "unblocked" s-t path in L_p with $\delta_x(i,j) > 0$ for all edges (i,j) on P. Mark all edges "blocked" which are examined but are not in P. If there is no "unblocked" s-t path GOTO Step 3. Otherwise proceed to Step 2.

Step 2 (augment) Calculate x' from x and the path P. $x \leftarrow x'$. GOTO Step 1.

Step 3 (stop) We have a maximal flow for this phase. STOP.

The point of this algorithm is that the augmentations made using L_p are exactly the augmentations made along lexicographically minimal shortest length s-t augmentation paths in H_x. First we will show that all the edges required are in L_p.

Theorem 4.1: If an augmenting path algorithm satisfies Properties 0 and 1, and augments along shortest length s-t augmenting paths then every edge that occurs in a shortest s-t augmenting path in phase p is in L_p.

Proof: Any node $i \in H$ that is on an s-t augmenting path P of length p during phase p satisfies $\sigma_{x'}(i) + \tau_{x'}(i) = p$ with respect to the current solution x'. By Property 1, $\sigma_x(i) + t_x(i) = p$, where x is the solution at the beginning of phase p. Then i is a node in L_p by Step 2 of Algorithm 4.2. Thus all nodes in P are in L_p. Moreover, $\sigma_x(i) = \sigma_{x'}(i)$ and $\tau_x(i) = t_{x'}(i)$, because if $\sigma(i)$ or $\tau(i)$ had increased, the other would have had to decrease, violating Property 1. (In other words, node i stays in the same layer.) Then by Step 3 in Algorithm 4.2, all edges in P are in L_p.

∎

Next we show that augmenting in L_p is equivalent to augmenting along lexicographically minimal shortest length s-t augmenting paths in H_x.

Theorem 4.2: If Properties 0-3 are satisfied, augmenting along lexicographically minimal shortest length s-t paths in H_x is equivalent to augmenting in L_p as given by Algorithm 4.3.

Proof: Let Q denote the path found by Algorithm 4.1 and let P denote the lexicographically minimal shortest length s-t augmenting path in H_x. By Step 1 of algorithm 4.3, Q is a shortest length s-t augmenting path. So $P \leq Q$.

Because of Theorem 4.1, P must correspond to a path in L_p. Now we have to prove that the arcs of P are not blocked. Let $\bar{x}$ be the initial feasible solution for this phase. We use induction on the number of blocked edges to prove a slightly stronger statement: none of the edges of any $\sigma_{\bar{x}}(v)$ length s-v path in H_x is blocked. Let (i,j) be a blocked edge and x be the feasible solution examined when (i,j) got blocked. By induction, the edges of the lex-min shortest s-j path in H_x were unblocked. Thus, as (i,j) got blocked, the lex-min $\sigma_{\bar{x}}(j)$ length s-j path in H_x went through an edge (i',j) with $i' < i$ in the ordering (if one exists). By property 3 there is not going to be any s-j path of length $\sigma_{\bar{x}}(j)$ later in this phase through the node i. Thus (i,k) is not on any s-v path of length $\sigma_{\bar{x}}(v)$.

Theorem 4.3: An augmenting path algorithm that satisfies Properties 0 through 3 and uses Algorithm 4.1 has time complexity $O(n^2ad+nau)$.

Proof: Setting up each L_p requires calculating $\delta_x(i,j)$ for every arc

and creating a network. So setting up all n L_p networks is $O(n \cdot a \cdot d)$. When solving the network, after n calculations of $\delta_x(i,j)$, either an edge can be marked "blocked" or an augmenting path is found. Since there are at most a edges in L_p, at most $O(n \cdot a \cdot d)$ time is needed to mark all edges blocked. There can be no more than a augmentations so $a \cdot u$ time is spent in a phase to update. Hence the total order is

$$O\big(nad + n\big(O(nad + au)\big)\big) = O(n^2ad + nau) \qquad \blacksquare$$

Since a must be $\Omega(n)$, the layered approach is at least as good as the non-layered method. Furthermore, as long as the update is not the most significant term, the layered algorithm will have a better order of magnitude.

In the following sections we will show that this improves the running time of three augmenting path algorithms.

5.0 Lawler and Martel's Polymatroidal Network Flow Algorithm

Lawler and Martel [11] address the problem of finding a maximum flow in a polymatroidal network. This problem, independently formulated by Hassin [9], is a powerful generalization of the classical maximum flow problem where, for each node, a polymatroid constrains the capacities of sets of arcs directed in to (and out of) the node. This structure also encompasses several classic matroidal problems, such as matroid or polymatroid intersection. In addition, it models some hitherto intractable scheduling problems [14].

Let G be a directed graph. The network has a distinguished source (sink) vertex s (t) with a virtual arc denoted by *(**). There are two polymatroids associated with each node of the network. They constrain

the flow on the arcs entering and leaving the node respectively (see [11] for details).

Lawler and Martel give an augmenting path algorithm for this problem. Their algorithm is generalization of Schönsleben's algorithm [15] for the polymatroid intersection problem. Their auxiliary digraph, H has all incident arc-node pairs as its node set. Lawler and Martel show that their algorithm satisfies properties 0 through 3 in Lemmas 6.1, 11.1, 9.1, and 11.3 respectively. To calculate the capacities they need a subroutine, which, given a polymatroid with a feasible function f and an element of the groundset e, calculates the maximum value δ such that f' is feasible, where $f'(e) = f(e)+\delta$ and $f'(e) = f(e')$ for all other edges e'. Let d denote the time complexity of such a subroutine for the polymatroids associated with the vertices of the graph.

Let m denote the number of arcs in the grah G. Then $n=O(m)$; $a=O(m^2)$; $u=O(m)$; $g=O(m)$. Theorem 4.3 gives an order of magnitude improvement over Lawler and Martel's $O(m^5 d)$ time bound.

Corollary 5.1. A maximum polymatroidal flow can be found in time $O(n^2 ad)$.

The details of the time complexity when using a different, "feasibility," subroutine are given in [16].

6.0 Frank's Algorithm

Edmonds and Giles [5] proved a general min-max relation pertaining to submodular functions on directed graphs. Their model includes a weighted version of Lawler-Martel's polymatroid network flows. (See [8]). It is more general, treating crossing submodular functions instead

of polymatroid rank functions. This min-max relation also includes the Lucchesi-Younger theorem [12] as a special case.

Let G be a directed graph with a crossing submodular function on the subsets of its nodes. This crossing submodular function constrains the net flow remaining in a set of vertices. Further, we have a linear cost function on the edges of G. The Edmonds-Giles problem is to find a feasible flow with minimum cost in this network. (See [8] for details.)

Frank [8] describes an augmenting path algorithm to find a feasible flow in this network. The algorithm is an essential part of Cunningham and Frank's [3] procedure for finding the minimum cost flow.

The complexity of the algorithm is $O(n^5h)$ where n is the cardinality of the groundset and h is the time required by a submodular function minimization oracle. We reduce the complexity to $O(n^4h)$. The auxiliary digraph, H and the capacities are given on page 225 in [8]. Theorem 1 and Case 1 on page 225 give Property 0; Lemma 5 and Lemma 7 state Properties 1 and 2 respectively. Property 3 follows from Lemma 6 and our Lemma 3.2. In Frank's algorithm the auxiliary digraph H has O(n) nodes; $a=O(n^2)$; d=h; u=O(n), where n is the number of nodes in G. Applying Theorem 4.3 gives the desired result.

Corollary 6.1. Algorithm 4.1 applied to Frank's algorithm produces an $O(n^4h)$ algorithm to find a feasible flow for the Edmonds-Giles problem.

7.0 Cunningham's Algorithm for Matroid Polyhedra

Cunningham [2] provides an algorithm to test membership in matroid polyhedra. Given a matroid M on a set E and a $y = (y_j : j \in E)$ the

membership problem is to determine whether y is in the convex hull of incidence vectors of independent sets of M.

A brief outline of Cunningham's method follows. Again the reader is referred to the original paper for details.

This algorithm works with solutions $x \leq y$ given in the form $x = \Sigma(\lambda_i I_i : i \in I)$ where $(I_i : i \in I)$ is a family of independent sets and λ_i for $i \in I$ are positive numbers such that $\Sigma(\lambda_i : i \in I) = 1$. It is an augmenting path algorithm. Augmenting paths are used to update x. The updated solution is better in the sense that $x \leq x' \leq y$ and $\Sigma(x_i : i \in E) < \Sigma(x'_i : i \in E)$. The algorithm starts with x=0.

The auxiliary digraph, H, has node set $N = E \{s,t\}$ and has no multiple edges. Cunningham has defined the lexical ordering from his r to s while we have defined the ordering in the opposite direction. Now reversing his auxiliary digraph, we get Properties 0, 1 and 2 from his Theorem 2.2, Lemma 3.2 and Lemma 4.5 respectively. Property 3 follows from Cunningham's Lemma 4.3 and our Lemma 3.2.

By our Theorem 4.2 it is possible to use a layered approach to Cunningham's algorithm. The layered algorithm, by Theorem 4.3 has at least as good a performance time bound.

Cunningham examines the complexity of a special case. Suppose M is given by the linear independence of the columns of an r by n matrix A. In this case it is possible to find the capacities for all edges quickly. Algorithm 4.1 needs the same time to calculate the capacities after each augmentation. It is possible, in time r^2n, to transform an independent set I_j in a form which allows such a quick calculation of the capacities. Therefore, Algorithm 4.1 has the same overall time complexity in this special case as Cunningham's algorithm: $O(r^2n^8)$. In other cases the

layered algorithm will have one order of magnitude better running time.

8.0 Further Parallels with Network Flows

The natural next extension to look for is an improvement on the $O(n^2ad+nau)$ time analagous to the algorithms of Karzonov [10], Cherkassky [1], and Malhotra, Pramodh Kumar and Maheshwari [13]. Here, however, the analogy to network flows breaks down. In Dinits' algorithm, the layered network is a genuine maximum network flow problem with its own capacities and flow constraints. Dinits gives a subroutine to find a maximum flow in this network; the improvements arise by replacing his subroutine with more efficient ones. In contrast, our graph L_p is not a genuine instance of the problem. Rather, it is an auxiliary digraph which serves as a useful bookkeeping device. It does not have the status of an independent subproblem.

The difficulty occurs with arcs in L_p that are not incident in L_p but interact together in the problem. This kind of interaction occurs in all three problems treated in Sections 5-7. To get Property 3, this problem is solved by lexicographic tie-breaking, a rule which is unnecessary in the classical case but crucial in these more complex cases. Thus there is less freedom to sequence augmentations during a phase, or to in effect perform several augmentations simultaneously (as in the MPM algorithm [13]).

In the case of maximum polymatroidal network flows, not only do non-incident arcs in L_p interact through the polymatroid constraints, but two distinct nodes in H_x may actually be edge-node pairs of the same edge (one for each node). Algorithm 4.1 does not explicitly prohibit both e_i and e_j from appearing in L_p. Now we show that this, in fact, cannot

occur. For this result we need to get deeper into the underlying structure of Lawler and Martel's algorithm. The reader is referred to [11] for the apropriate definitions.

Theorem 8.1. If $e = (i,j)$ then e_i and e_j cannot both appear in the layered graph L_p when algorithm 4.1 is applied to the polymatroid network flow problem.

Proof: In other words, the edge e cannot appear as both forwards and backwards arc in the set of all shortest augmenting paths with respect to some flow f. The proof is by contradiction.

By assumption $(s,P_1,i,j,P_2,t) \equiv Q^1$

and $(s,P_3,j,i,P_4,t) \equiv Q^2$

are both minimal length augmenting paths, where $P_1(P_3)$ is an s-i(s-j) path and $P_2(P_4)$ is a j-t(i-t) path (see Figure 1). Consider the two paths (s,P_1,P_4,t) and (s,P_3,P_2,t) and let the latter be shorter. Then this path (call it P) is strictly shorter than Q^1 and Q^2 and hence is not augmenting. If we let e_1 be the last arc of P_3 and e_2 the first arc of P_2 then (e_1,e) and (e,e_2) are admissible but (e_1,e_2) is not.

Now why is (e_1,e_2) not admissible? Either because e_1 is a forward arc on P with a saturated head and $e_2 \varepsilon H(e_1)$; or else e_2 is a forward arc on P with a saturated tail and $e_1 \varepsilon T(e_2)$.

First Case: e_1 is a forward arc with a saturated head and $e_2 \varepsilon H(e_1)$.

Since (e_1,e) is admissible we must have $e \varepsilon H(e_1)$, thus e is directed from i to j. Moreover the arc e has saturated head and $H(e) \_ H(e_1)$, thus $e_2 \varepsilon H(e)$. This contradicts the assumption that (e,e_2) is

admissible.

Second Case: e_2 is a forward arc with saturated tail and $e_1 \varepsilon T(e_2)$.

The proof is similar to the one in the first case.

References

1. Cherkassky, B., "Efficient algorithms for the maximum flow problem," Akad. Nauk USSR, CEMI, Mathematical Methods for the Solution of Economical Problems, Vol. 7, 1977, pp. 117-126.

2. Cunningham, H. W., "Testing membership in matroid polyhedra." J. Combin Theory Ser. B 36 (1984), pp. 161-188.

3. Cunningham, H. W. and A. Frank, "A primal-dual algorithm for submodular flows," Math. Oper. Res., in press.

4. Dinits, E. A., "Algorithm for solution of a problem of maximum flow in a network with power estimation," Soviet Mth. Dokl. 11(1970), pp. 1277-1280.

5. Edmond, J. and R. Giles, "A min-max relation for submodular functions on grahs," in "Studies in Integer Programming Proceedings, Workshop on Programming, Bonn, 1975," (P. L. Hammer et al. eds.), Annals of Discrete Math. No. 1, pp. 185-204, North Holland, Amsterdam, 1977.

6. Edmonds, J. and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," J. Assoc. Comput. Mach. 19 (1972), pp. 248-264.

7. Ford, L. R. and D. R. Fulkerson, "Flows in Networks," Princeton Univ. Press, Princeton, N.J., 1962.

8. Frank, A., "Finding feasible vectors of Edmonds-Giles polyhedra," J. Combin. Theory Ser. B 36, pp. 221-239.

9. Hassin, R., "On network flows," Network 12, (1981), pp. 1-12.

10. Karzonov, A. V., "Determining the maximum flow in a network by the method of preflows," Soviet Math. Dokl., Vol. 15, 1974, pp. 434-437.

11. Lawler, E. L. and C. U. Martel, "Computing maximal polymatroidal network flows," Math. Oper. Res. 7, No. 3 (1982).

12. Lucchesi, C. and D. H. Younger, "A minimax theorem for directed graphs," J. London Math. Soc. 17 (2), (1978), pp. 369-374.

13. Malhotra, V. M., M. Pramodh Kumar and S. N. Maheshwari, "An $O(|V|^3)$ algorithm for finding maximum flows in networks," Computer Science Program, Indian Institute of Technology, Kanpur 208016, India, 1978.

14. Martel, C. U., "Generalized Network Flows with an application to multiprocessor scheduling," Ph.D. Thesis, University of California, Berkeley, 1980.

15. Schönsleben, P., "Ganzzahlige polymatroid-intersection-algorithmen," Ph.D. Thesis, Eidgenössischen Technischen Hochschule, Zürich, 1980.

16. Tovey, C. A. and M. A. Trick, "An $O(m^4 d)$ Algorithm for Polymatroid Flows," Georgia Institute of Technology, ISyE Report J83-9, 1983.

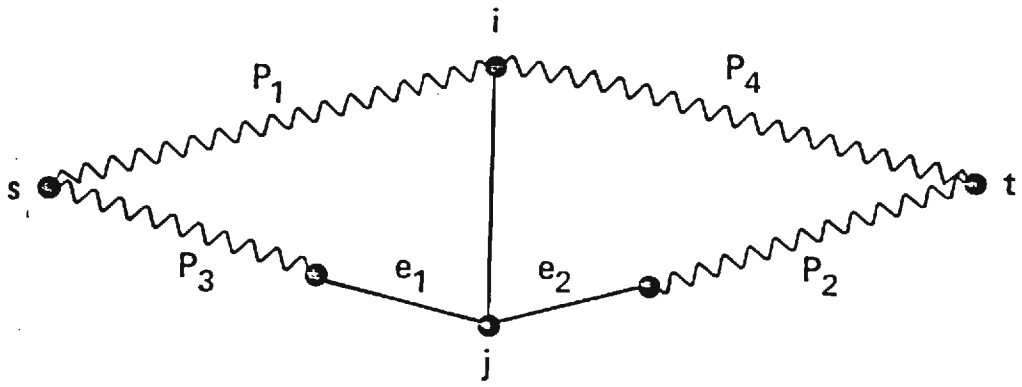


Figure 1